SHAASTRA2022

Centre For Innovation

BUILD YOUR OWN

AUTONOMOUS ROBOT

SHAASTRA 2022
SESSION 1

SHAASTRA2022™

Centre For Innovation

What's in store?

- Introduction to Robotics
- Getting Started with CoppeliaSim
- Programming in Lua
- Building a Robot in Coppeliasim

SHAASTRA2022™

Centre For Innovation

01

What is Robotics?

SHAASTRA2022

Centre For Innovation

Introduction

Robotics is the intersection of science, engineering and technology that produces machines, called **robots**, that substitute for (or replicate) human actions.

SHAASTRA2022

Centre For Innovation

Guess Who?

SHAASTRA2022

Centre For Innovation

Types of Robots

Generally, there are 5 types of robots;

Pre-programmed Robots:

- Operate in controlled environments
- Simple and monotonous tasks
- Example: Mechanical hand in industries

Humanoid Robots:

- Mimic human behavior
- Perform activities like running, jumping and carrying objects
- Example: Sophia and Atlas

Autonomous Robots:

- Have the ability to take their own decisions
- Work independently of human operators
- Example: Roomba

Teleoperated Robots:

- Semi-autonomous robots
- Use wireless networks to enable human control from safe distances
- Example: Drones to detect Landmines

Augmenting Robots:

- Can enhance current human capabilities
- Replace the capabilities a human may have lost
- Example: Robotic Prosthetic limbs

SHAASTRA2022™

Centre For Innovation

Humans vs Robots

Humans	Robots
Sense Organs	Sensors
Brain	Control System
Muscles	Actuators
Food	Power Supply

SHAASTRA2022™

Centre For Innovation

APPLICATIONS

- Helping fight forest fires
- Working alongside humans in manufacturing plants (known as co-bots)
- Robots that offer companionship to elderly individuals
- Surgical assistants
- Last-mile package and food order delivery
- Autonomous household robots that carry out tasks like vacuuming and mowing the grass
- Assisting with finding items and carrying them throughout warehouses
- Used during search-and-rescue missions after natural disasters
- Landmine detectors in war zones

SHAASTRA2022™

Centre For Innovation

02

Getting Started with CoppeliaSim

Let's simulate your own world..!

SHAASTRA2022

Centre For Innovation

What is CoppeliaSim ...?

The robot simulator CoppeliaSim is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a remote API client, or a custom solution. This makes CoppeliaSim very versatile and ideal for multi-robot applications.

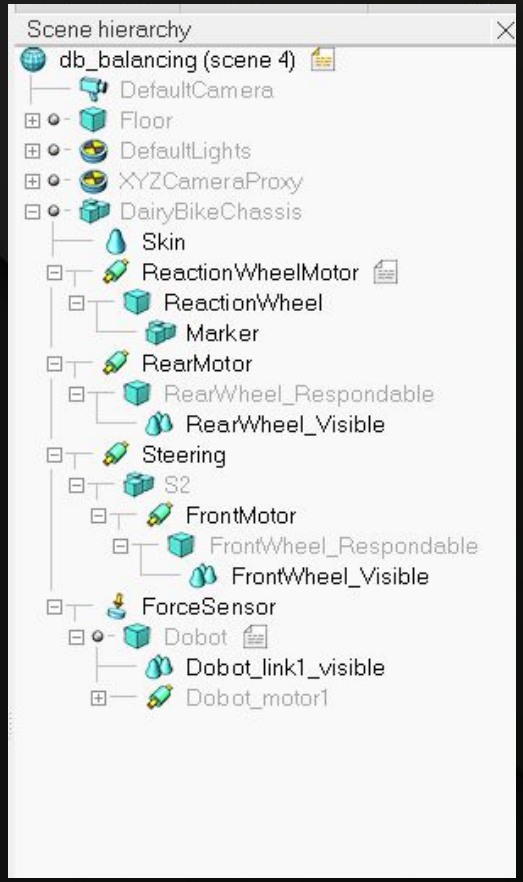
The documentation for Coppeliasim is here.

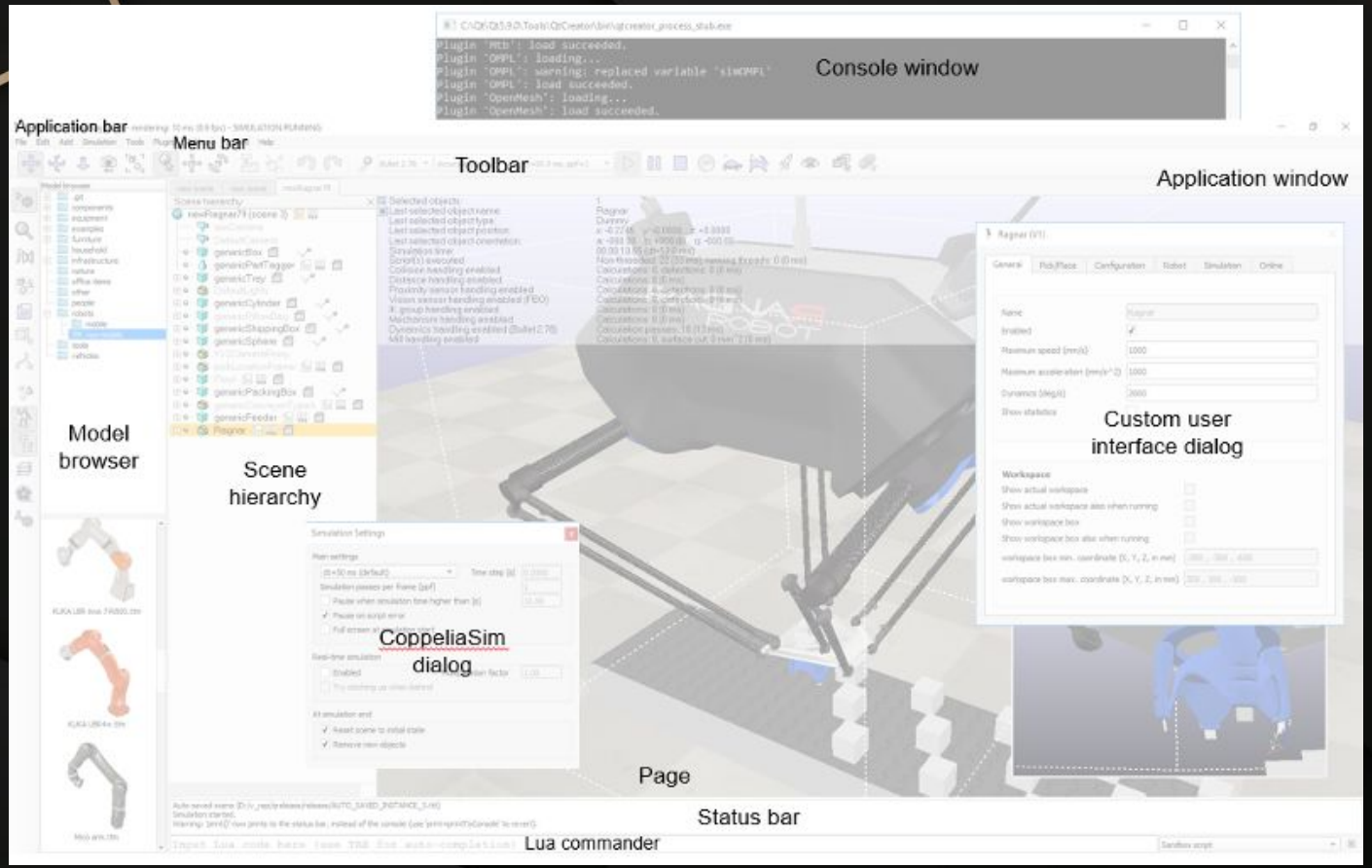
SHAASTRA2022

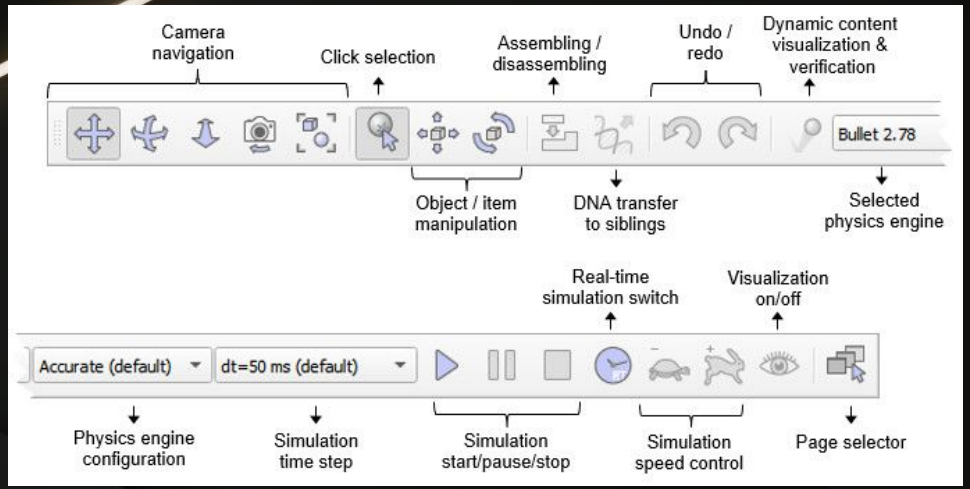
Centre For Innovation

Terminology

- Scene
- Models
- Model Browser
- Scene Hierarchy
  - Parent
  - Child
- Objects
- Scripts
  - Threaded
  - Non-threaded
- Menu bar
- Toolbar
- Visibility Layers




Console window
Application bar
Menu bar
Toolbar
Application window
Model browser
Scene hierarchy
Custom user interface dialog
CoppeliaSim dialog
Page
Status bar
Lua commander



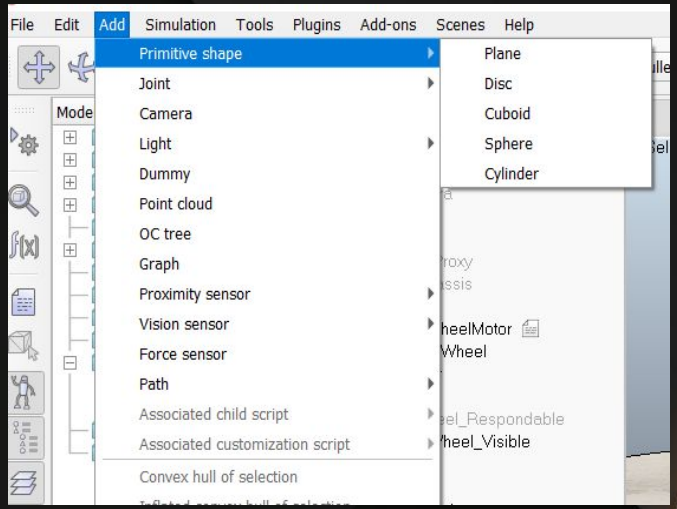
- Toolbar
  - Pan tools
  - Position manipulation
  - Orientation manipulation
  - Undo-Redo
  - Physics Engines
    - Bullet 2.78
    - Bullet 2.83
    - ODE
    - Vortex
    - Newton
  - Simulation control
  - Page Selector

SHAASTRA2022

Centre For Innovation

Menu bar

- Add Menu
  - Primitive Shapes - Plane, Disc, Cuboid, Cylinder and Sphere
  - Joints - Revolute, Prismatic and Spherical
  - Dummy
  - Graph
  - Sensors
    - Proximity Sensors
    - Vision Sensors
    - Force Sensors
  - Paths



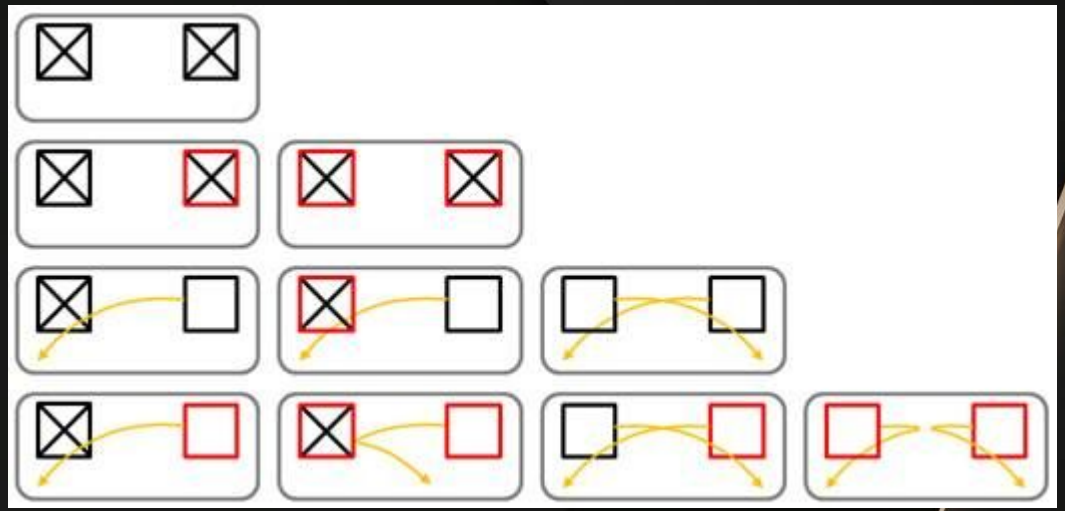
SHAASTRA2022

Centre For Innovation

Object Properties

- Colour
- Geometry/Size
- Respondable and Non-Respondable
- Dynamic and Static

	Static	Non-static
Non-respondable	☒	☐
Respondable	☒	☐



Special Properties:

- **Collidable** - allows enabling or disabling collision detection capability for the selected collidable object.
- **Detectable** - allows enabling or disabling proximity sensor detection capability for the selected detectable object. Clicking details allows you to edit the detectable details.
- **Renderable** - allows enabling or disabling the vision sensor detection capability for the selected renderable object.
- **Measurable** - allows enabling or disabling minimum distance calculation capability for the selected measurable object.

SHAASTRA2022

Centre For Innovation

Lua Programming

What is Lua?
Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

- Easy to use
- Flexible
- Portable
- Powerful
- Embeddable and Small
- Interpreter

Applications: Used in Embedded Systems and Gaming

SHAASTRA2022

Centre For Innovation

Keywords

and	break	do	else
elseif	end	false	for
function	if	in	local
nil	not	or	repeat
return	then	true	until
while			

Data types

- **nil** - Used to differentiate the value from having some data or no(nil) data.
- **boolean -** Includes true and false as values. Generally used for condition checking.
- **number -** Represents real(double precision floating point) numbers.
- **String** - Represents array of characters.
- **Function** - Represents a method that is written in C or Lua.
- **table** - Represent ordinary arrays, symbol tables, sets, records, graphs, trees, etc., and implements associative arrays. It can hold any value (except nil).

Type function is used to determine the datatype of the variable. Example:

```
print(type("What is my type"))    --> string
```

SHAASTRA2022

Centre For Innovation

Operators

An operator is a symbol that tells the interpreter to perform specific mathematical or logical manipulations.

- Arithmetic Operators - Example: +, -, *, /, %
- Relational Operators - Example: ==, ~=, >, <, >=, <=
- Logical Operators - Example: and, or, not
- Misc Operators
  - .. - Concatenates two strings
  - # - Returns the length of a string or table

SHAASTRA2022

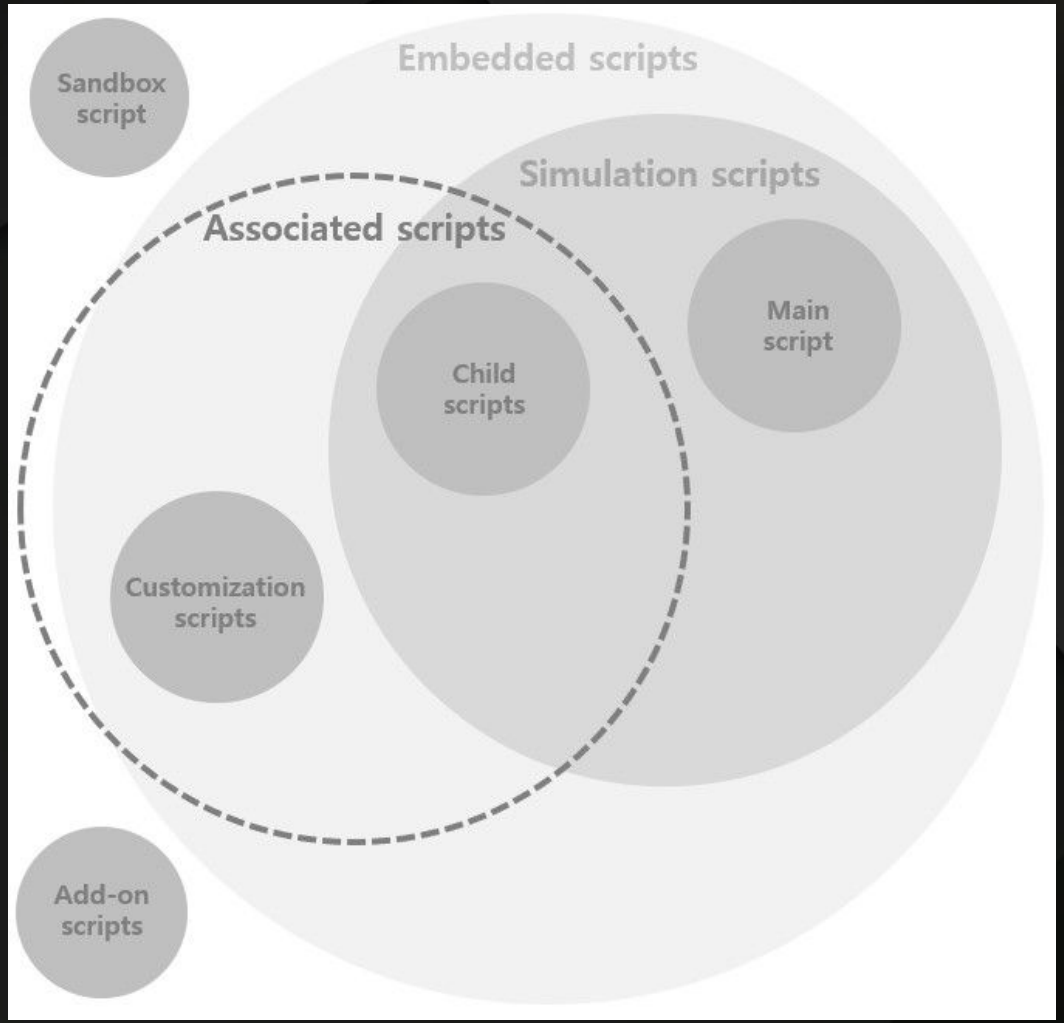
Programming in Coppeliasim

Centre For Innovation

- Script
- Main Script
- Child Script
  - Threaded
  - Non-threaded
- Plugins
- Sandbox Script/ Add-ons

Programming Languages which can be used in Coppeliasim:

- Lua
- C/C++
- Python
- Java
- Matlab
- Octave


Embedded scripts
Sandbox script
Simulation scripts
Associated scripts
Main script
Child scripts
Customization scripts
Add-on scripts

Coroutines:
Coroutines are computer program components that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed.

Threaded Function:
Threading in a script is achieved via coroutines, that CoppeliaSim will preemptively interrupt (i.e. yield, or switch) at regular intervals, and resume at a later point.

Non-Threaded Function:
The entrance to a script is always originating from callback functions which are not running threaded, and should not be blocking. This means that every time they are called, they should perform some task and then return control. If control is not returned, then CoppeliaSim will halt.

Note: Threaded code has several weaknesses compared to non-threaded code if not programmed appropriately: it is more resource-intensive, and it can waste some processing time.

SHAASTRA2022

Centre For Innovation

Callback Functions

- **sysCall_init -** Initialization function. This callback function is not optional. It will be executed one time just at the beginning of a simulation.
- **sysCall_actuation -** the actuation function. This callback function will be executed in each simulation pass. The code is in charge of handling all the actuation functionality of the simulation.
- **sysCall_cleanup -** the restoration function. This function will be executed one time just before a simulation ends. The code is in charge of restoring object's initial configuration, clearing sensor states, etc.
- **sysCall_sensing -** the sensing function. This callback function will be executed in each simulation pass. The code is in charge of handling all the sensing functionality of the simulation (proximity sensors, etc.) in a generic way

Basic APIs

1. sim.getObjectHandle(): Retrieves an object handle based on its name.
   Syntax: int objectHandle=sim.getObjectHandle(string/int objectName)

2. sim.setJointPosition(): Sets the intrinsic position of a joint. May have no effect depending on the joint mode. This function cannot be used with spherical joints
   Syntax: sim.setJointPosition(int objectHandle,float position)

3. sim.getJointPosition(): Retrieves the intrinsic position of a joint. This function cannot be used with spherical joints.
   Syntax: float position=sim.getJointPosition(int objectHandle)

SHAASTRA2022™

Centre For Innovation

Work out Time!

Let's build a bot from scratch!

SHAASTRA2022

Centre For Innovation

SHAASTRA2022

Centre For Innovation

THANK YOU

Aneesh Kandi
2nd Year Undergraduate
ee20b009@smail.iitm.ac.in

Sreehari M
2nd Year Undergraduate
me20b170@smail.iitm.ac.in